Software-Defined Multipath-TCP for Smart Mobile Devices

Tom De Schepper, Jakob Struye, Ensar Zeljković, Steven Latré and Jeroen Famaey
University of Antwerp - imec, IDLab, Department of Mathematics and Computer Science, Belgium
firstname.lastname@uantwerpen.be

***Abstract*—Current mobile consumer devices are equipped with the ability to connect to the Internet using a variety of heterogeneous wireless network technologies (e.g., Wi-Fi and LTE). These devices generally opt to statically connect using a single technology, based on predefined priorities. This static behavior does not allow the network to unlock its full potential, which becomes increasingly more important as the requirements of services, in terms of for example throughput and reliability, grow. Multipath TCP (MPTCP) is a solution that allows the simultaneous use of multiple network interfaces. However, it does this uncoordinated for a single connection between two endpoints. Therefore, this paper proposes a Software-Defined Networking (SDN) architecture to enable coordinated multi-path routing across the several networks for mobile devices. Moreover, we propose a novel weighted MPTCP scheduler that allows the transmission of certain controllable percentages of data per network interface. The proposed idea is evaluated through a real-life prototype implementation with a smartphone.**

***Index Terms*—SDN, heterogeneous networks, Multipath TCP**

I. Introduction

Today's mobile consumer devices are equipped with the ability to connect to the Internet using a variety of different heterogeneous wireless network technologies (e.g., Wi-Fi, LTE, and Bluetooth). Over the next few years, the diversity among devices and technologies is expected to expand further with, for instance, the rise of all kinds of (possibly mission critical) Internet of Things (IoT) devices and multimedia services, as well as the availability of new technologies such as sub-1GHz Wi-Fi (i.e., IEEE 802.11ah), 60 GHz Wi-Fi (i.e., IEEE 802.11ad) and visible light communications (e.g., Light Fidelity (Li-Fi)). These devices and applications will have stringent and diverse quality requirements: for instance, high throughput for high quality video or virtual reality applications, or the reliable delivery of sensor data for critical infrastructure, such as energy supply, in buildings.

Current wireless and mobile devices are generally managed in a static way. They select one of the available network interfaces to connect to the Internet based on predefined priorities (e.g., 5 GHz Wi-Fi if available, otherwise 2.4 GHz Wi-Fi, otherwise LTE). In some cases, the user can manually override these priorities, but this is not user friendly, as it is not done in a transparent manner. The abstraction from network connectivity is a key aspect in terms of user friendliness and Quality of Service (QoS). Furthermore, devices select only one interface, while the simultaneous use of multiple networks can help to increase the available bandwidth and reliability.

In early 2013, the Multipath TCP (MPTCP) standard was released as an extension to regular TCP [1]. This extension enables the transmission and reception of data concurrently on multiple network interfaces. Multiple regular TCP connections, denoted as subflows, are combined into a single MPTCP connection, while each subflow can follow a different path. The division of data among the different subflows is decided by a scheduler. MPTCP is most often used in smartphones, where it combines the Wi-Fi and cellular networks [2]. For instance, Apple uses MPTCP in their digital assistant Siri. While MPTCP aims to improve QoS and network resource utilization, it only focuses on the alternative paths between two hosts and not on a network-wide scale.

To this extent, this paper proposes a Software-Defined Networking (SDN) framework that allows for coordinated multipath routing across several managed networks. A centralized controller provides a specific configuration to each MPTCP-enabled device under its control. On these devices there is an interface that is capable of receiving the configuration and applying it. In more detail, the contributions of this paper are fourfold. First, we present a novel Weighted Round-Robin (WRR) scheduler for MPTCP that is capable of dividing traffic across different network interfaces or MPTCP subflows, based on weights per interface. It thus becomes possible to perform per-device load balancing on a packet level. Second, we propose an SDN-based architecture where a centralized controller can dynamically adapt the parameters of each WRR scheduler across all devices in the network. This allows for a dynamic and coordinated way of multipath routing for heterogeneous networks and offers increased throughput and reliability, essential to modern applications. Third, we describe a real-life prototype and use that to evaluate our novel scheduler. Fourth, we provide the, to our knowledge, first port of MPTCP to Android 6.0 (Marshmallow). This port of MPTCP is publicly available on Github[1].

The remainder of this paper is structured as follows. We start by giving an overview of the current state of the art in Section II. Next, we present our framework in Section III. Section IV discusses the prototype, while that prototype is used to evaluate the scheduler in Section V. Finally, conclusions are provided in Section VI.

[1] https://github.com/imec-idlab/Multipath-TCP-Android6.0-Marshmallow

978-3-901882-98-2

II. Related work

We propose an SDN-based framework for coordinated MPTCP use in heterogeneous wireless networks and present a novel MPTCP scheduler. In other words, this research is situated on the intersection of two active research domains, namely MPTCP and SDN (in wireless networks). We describe the state-of-the-art in both domains.

A. Multipath-TCP scheduling

MPTCP is an extension to regular TCP aimed at multi-homed devices such as smartphones (with Wi-Fi and mobile interfaces) or servers (with multiple Ethernet interfaces) [1]. It maintains separate data paths on multiple interfaces, combining these subflows into one logical connection. Application data can be divided across these subflows to attain a higher throughput, or duplicated for reliability. Additionally, one subflow could be kept idle and only used when the main subflow is broken. In this case the fallback subflow is already established, meaning the handover can occur very quickly. Moreover, higher layers still see the same MPTCP connection, meaning the handover was fully transparent to those layers. As MPTCP is fully backwards-compatible with regular TCP, an MPTCP-aware host attempts to use MPTCP when establishing a new connection, but falls back to regular TCP gracefully when the other endpoint does not indicate it is MPTCP-aware.

A key component in the MPTCP implementation is the scheduler that must decide upon which subflow(s) to send each TCP segment. In the default MPTCP implementation there are three schedulers present [3]. First, Lowest Round Trip Time First (LowRTT) scheduler is the default option and selects the subflow with the lowest RTT when a segment is to be scheduled. Once a subflow is chosen, all following segments are also sent using that subflow, until its congestion window is filled. Second, Round-Robin (RR) is a fairly simple scheduler rotating through all available subflows. This scheduler only sends a fixed number of segments, denoted by a parameter that is by default 1, on a subflow before continuing to the next subflow. Third, the Redundant scheduler sends every segment over every available subflow and offers increased reliability.

Due to its importance, quite some research has already been done towards improving the scheduler. The modular scheduler system was originally presented by Paasch et al. [3]. A thorough evaluation of the default LowRTT and RR schedulers shows that LowRTT vastly outperforms the RR scheduler [3, 4]. Specifically, in terms of throughput there is a difference of up to 33 %, while in terms of delay-jitter the LowRTT scheduler shows in 80 % of the conducted experiments a delay-increase of under 100 %, compared to only 40 % for the RR approach. Furthermore, it is shown that even a random scheduler consistently outperforms the RR scheduler, in terms of throughput, by up to 12.5 % [4].

Multiple alternative schedulers have been proposed as well. Yang et al. propose an alternative scheduler that chooses subflows based on an estimation of how much more traffic they can handle before becoming congested [5]. The scheduler, which requires some configuration, is shown to attain a throughput of up to 14.5 % higher than the default LowRTT scheduler. Furthermore, an alternative scheduling strategy aimed at optimizing latency for short-lived connections is presented [6]. When one subflow's RTT is a lot higher than another subflow's, the scheduler will not use the slower subflow at all when a connection was just established. This allows for a transmission of a flow (of 384 KB) that is almost twice as fast as the default LowRTT scheduler. Finally, the BLocking ESTimation (BLEST) scheduler is proposed that aims to reduce Head-of-line (Hol) blocking due to packet reordering [7]. This is the phenomenon where segments sent over a low-delay subflow cannot be processed at the receiver until preceding segments sent over a high-delay subflow are received. The BLEST scheduler estimates the risk of HoL-blocking before sending segments over a considered subflow, and skips the subflow if the risk is deemed too high.

To conclude our discussion of MPTCP, we note that recently different evaluations of the impact of MPTCP in real-life scenarios, especially with smartphones, have been conducted [8, 9]. Improvements in throughput and reliability are shown but at the cost of an increased energy consumption.

B. Software-defined wireless networks

Recently some proposals has been made to move SDN techniques into the wireless domain, in particular wireless home networks [10, 11]. In these architectures a centralized controller is used to cope with the large differences among (smart) devices and user requirements in the home network. This supports functionalities such as home automation based on the user's location, configurable lifestyle management and condition monitoring [10]. Furthermore, it would be possible for a controller to dynamically and directly program networking devices and configure their MAC policies [11].

The most concrete approaches rely often on an OpenFlow (OF) controller (e.g. Ryu) that uses the OF communication protocol to configure virtualized switches (e.g., Open vSwitch (OVS)) [12, 13]. The virtualized switch reports in real-time monitored flow information (e.g., counter for the number of packets and bytes) to the controller by making use of OF stats request and reply messages. This monitoring information can be used at the control-side to perform network optimizations. An example of such a optimization is the performance of inter-technology handovers en load balancing [13]. Another solution worth mentioning, is the 5G-EmPOWER framework that focuses on virtualized network functions in wireless networks [14]. In line with the thought of Network Function Virtualization (NFV), it moves intelligence from an access point (AP) to a controller. Currently they focus on the following control aspects: wireless clients state management, resource allocation, network monitoring, and network reconfiguration.

To summarize, our presented work combines two interesting research areas: namely MPTCP and SDN in local area networks (LANs). Most work in the domain of MPTCP has been done towards the evaluation of the default schedulers and the development of additional schedulers. While MPTCP enables increases in throughput and reliability, it only focuses on the

alternative paths between two hosts and not on a network-wide scale. Furthermore, we can report that SDN techniques are being introduced in wireless environments. Mostly OF-based solutions are applied to interact with virtualized switches. This contrasts our solution, where a centralized controller interacts directly with an MPTCP scheduler on consumer devices.

III. SDN-based coordinated MPTCP framework

In this section we describe the proposed framework to support coordinated MPTCP in heterogeneous wireless networks. We first introduce a novel WRR scheduler for MPTCP that enables the configurable load balancing of TCP segments across multiple interfaces. Afterwards we describe the entire architecture of the framework where a centralized controller can dynamically reconfigure the settings of WRR schedulers across different devices in the network.

A. Weighted Round-Robin scheduler

While many alternative MPTCP schedulers, optimizing different metrics, have been proposed and implemented, selection of subflows is mostly based on lowest RTT or tries to cope with different latencies across different subflows and paths through the network. These schedulers operate in an uncoordinated way and do not take other traffic streams into account. For instance, it might be possible that a Wi-Fi network is already being used by a large number of stations, but the MPTCP scheduler will still opt to use this interface, which might lead to packet loss. Therefore, we propose a novel WRR scheduler that enables load balancing across different subflows or interfaces based on configurable weights. The regular RR scheduler, included in the standard MPTCP implementation, already has a parameter *num_segments* that controls how many segments are sent per subflow before moving on to the next in one round-robin iteration. In our WRR scheduler, which is based on the original RR scheduler, this *num_segments* parameter can be set separately for each subflow. Currently, subflows are identified based on IP addresses. As a result, multiple subflows sharing a source address also share a *num_segments* limit. This is desirable as MPTCP creates multiple subflows on a certain interface and a separate *num_segments* value should not be set for each subflow, but per network interface.

Furthermore, another modification to the standard RR scheduler was needed. While the RR scheduler starts a new round if each available subflow has reached its *num_segments*, the WRR scheduler only starts a new round if all subflows, including unavailable ones (like due to a full congestion window), have reached the *num_segments* limit. If all subflows that have not yet hit the limit are unavailable, the scheduler will not schedule any segments until one becomes available. While this could have a negative effect on throughput when RTT varies significantly from one subflow to another, it is necessary to ensure the relative *num_segments* sizes are respected across the entire MPTCP session.

Finally, we would like to point out that while the configuration of the *num_segments* values per network interface can be done by a centralized controller, it is also possible to do this individually on a device-level. This can, for instance, be interesting for a scenario where a user wants to limit the monetary cost of certain technologies, such as cellular technologies (e.g., LTE). This can be done by setting the weights appropriately.

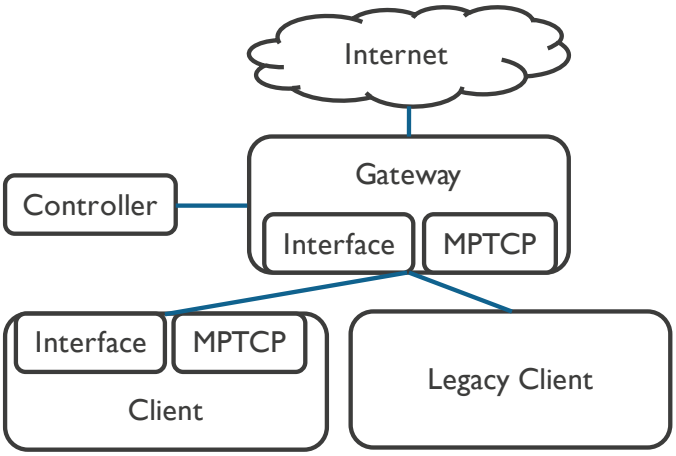


Fig. 1: Schematic overview of the framework

B. Architectural overview

In the previous section we have explained the proposed WRR scheduler for MPTCP but we envision a larger SDN-based framework that applies this scheduler and MPTCP in a coordinated fashion. Figure 1 shows the relevant components of the architecture. The architecture consists of an SDN controller, a gateway and a multitude of clients. In an ideal scenario, we require the deployment of MPTCP on the gateway and all clients, allowing for a fully coordinated use of MPTCP in the network. All MPTCP-enabled devices require a convenient manner to configure the weights for the WRR scheduler. This is an interface that can communicate with the controller to receive its device-specific configuration. For instance, on an Android device, this interface can be a new app or an extension of the required MPTCP app. The controller broadcasts its address to all devices in the network, to setup the communication. For the communication, the publish-subscribe pattern is most suited, because of the needed one-way communication from controller to all clients to update their configuration. For the definition of the configuration we propose the YANG data model language as it offers a simple and standardized manner of accessing and updating the data.

Not all networks will only contain MPTCP-enabled devices. Our presented architecture supports these different topologies as well, but consequently only a limited level of control is possible. If the gateway is not MPTCP-enabled, it is still possible to configure client devices that do have the required interface and MPTCP implementation. These devices can communicate with services on the Internet that do support MPTCP, so by configuring the weights on the client device, upstream traffic can still be controlled. It is also possible that a client device is MPTCP-enabled but does not contain the required interface to communicate with our controller. In such

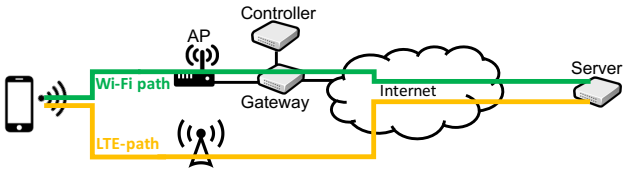


Fig. 2: Overview of real-life prototype

a case, it is still possible to load balance the downstream traffic by modifying the weights on the gateway. Furthermore, as already mentioned in the previous section, it is possible to use the scheduler in a stand-alone manner without a controller and letting the device or user itself decide on the configuration.

Finally, we also like to point out that our described architecture only contains the elements to configure and communicate with MPTCP-enabled devices. On top of this framework, logic needs to be implemented that decides on how to configure all the devices in the network in order to improve, for instance, network-wide throughput. The development of such algorithms or intelligence is out of the scope of this paper. Moreover, we acknowledge the fact that to be able to make such decisions, the current network state needs to be known. For the deployment and use of such a monitoring framework we refer to existing work, discussed in Section II.

IV. Implementation and prototype description

One of the most interesting use-cases of MPTCP is its deployment in smartphones, because of, among others, the absence of a dedicated network connection. We thus opt for a prototype with Android smartphones as clients. Different network interfaces and technologies are needed to use MPTCP. Within the scope of smartphones, the two most relevant options for technologies are Wi-Fi and LTE. To support public LTE networks, the MPTCP server is placed in the public Internet instead of on the home gateway. In this setup it is possible for our controller to configure both endpoints of the MPTCP stream, namely on the server and client. For the communication between the controller and the different devices, we make us of the ZeroMQ framework. This framework allows for distributed, reliable and scalable communication between different platforms. On all devices, we implement the required interface for the communication and capable of handling ZeroMQ sockets. This setup allows us to evaluate our WRR scheduler in the next section. The following hardware is used: as a smartphone we use the Motorola Moto G 4G ('Peregrine') with a port of MPTCP to Android 6.0 (Marshmallow) (as the most recent Android port of MPTCP, to the best of our knowledge, was based on Android 4.4.4 (Kitkat)). Furthermore, a Netgear AC1900 wireless router with 802.11n is positioned in the same room as the smartphone with line of sight. As a controller we use an Intel NUC and the MPTCP server is running on a virtual machine, roughly 160 km away. The following network subscriptions are used: Telenet Business 200 for Wi-Fi (upload of 25 Mpbs) and Mobile Vikings data-only SIM card for LTE connection. The setup is illustrated by Figure 2.

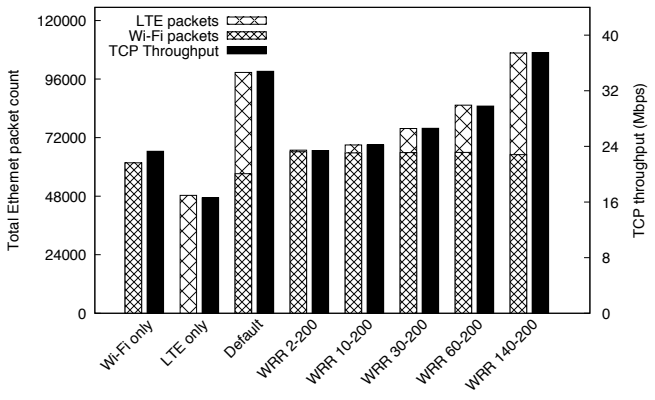
V. Evaluation and discussion

We focus on the evaluation of the novel WRR MPTCP scheduler as this is a key component of our framework. The most important behavior that requires verification, is how correctly the assigned weights are respected, as that is key to be able to perform the correct load balancing across different interfaces. Furthermore, we investigate the impact of our scheduler with different weights in terms of throughput, number of packets and RTT, and compare it to cases where only one interface is used (i.e., regular TCP) and the default LowRTT MPTCP scheduler. For this evaluation we make use of the previously described prototype and set weights on the smartphone by instructions from the controller and average results across 15 runs for each configuration. Each separate test consists of a 30 second iPerf TCP connection from the phone to the server, in which the phone sends as many TCP segments as possible. For the scheduler, we consider the weight for Wi-Fi to be on 200, while we vary the number of segments of LTE between 2, 10, 30, 60 and 140.

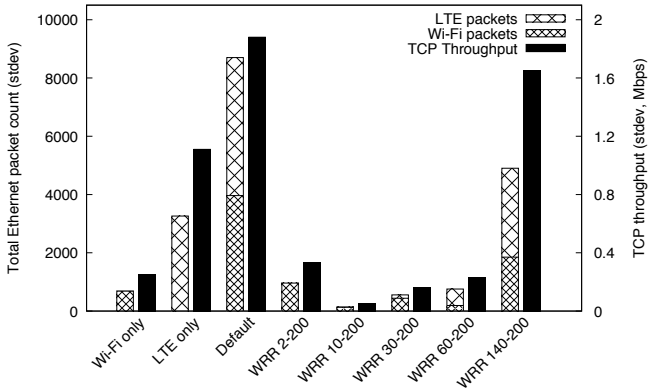
A. WRR scheduler validation

Figure 3a shows the median measured throughput and packet counts for all experiments, and Figure 3b shows the standard deviations of those measurements. With only Wi-Fi, a median throughput of 23.3 Mbps is measured, close to the theoretical 25 Mbps capacity, and 61667 Ethernet packets are received at the server. For pure LTE a median throughput of 16.6 Mbps was achieved at 48344 packets. The packets to throughput ratio is higher than in the previous test, indication of some retransmissions as the payload size was equal across both tests. Additionally the standard deviations for LTE are considerably higher than for Wi-Fi, indicating a slightly less stable cellular network performance. With both interfaces enabled, the default MPTCP scheduler reaches a median throughput of 34.8 Mbps. While higher than either network's separate throughput, it is lower than the sum of the two. A median of 41.4 % of packets was sent over the cellular network. The percentage of traffic sent over the slower network can vary significantly between repetitions, as it is based on the measured RTTs, which explains the high standard deviation.

For all WRR tests, the Wi-Fi packet count is counter-intuitively higher than in the Wi-Fi only test. Analysis of the packet flows in Wireshark showed that packets in the WRR tests consistently dropped from 1514 bytes to 1414 bytes after 0.5 s. It is unclear why this happens, but this does not effect the actual TCP throughput and is thus ignored. Next we observe that there are no noticeable differences in Wi-Fi packet count between the different WRR configurations. In these experiments the throughput of the network subscriptions is the bottleneck on the total TCP throughput. While the phone is capable of sending more packets over the Wi-Fi interface, it is being throttled by the network provider. With the WRR scheduler, the cellular network can, in a sense, fill some of



(a) Throughput and packet counts



(b) Standard deviations

Fig. 3: TCP throughput and received packet counts with standard deviation for different experiments

the gap left by the bandwidth throttling. Note that, within one cycle of round-robin, the WRR scheduler always prefers a subflow that has sent at least one packet over a subflow that has not sent anything yet. Assuming the Wi-Fi subflow is chosen first, the scheduler will keep using that subflow until either its weight is reached, or it becomes unavailable (e.g. due to a full TCP window). In this second case, the kernel would not be able to immediately send another packet in an MPTCP connection with only one subflow. However, due to the WRR scheduler, more packets can be sent over the LTE subflow at this point. This scenario explains why the Wi-Fi packet count is stable through all WRR tests.

Next we analyze how closely the set weights are respected. The LTE weight is always chosen as a percentage of the fixed Wi-Fi weight. With weights of 2 and 200, 0.99 % of all packets should theoretically be sent over the LTE subflow. Measurements show a median of 0.9 % of all packets on the LTE subflow. With LTE weight 10, the theoretical share increases to 4.8 %, with a measured median share of 4.7 %. For weight 30, theoretical and measured shares are both 13.0 %. while, for weight 60, these are 23.1 % and 22.8 % respectively. For the final experiment the LTE connection has a theoretical share of 41.2 % of all packets with a measured median share

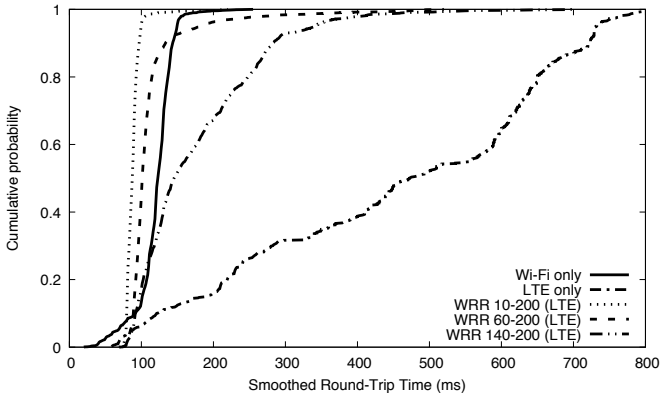


Fig. 4: Smoothed round-trip time for different scenarios

of 38.9 % These results clearly show that the configured LTE weights are approached closely and never exceeded. It makes sense that the measured share is sometimes slightly smaller than the theoretical one: the Wi-Fi subflow is established first, giving that subflow a slight head start. Additionally we noticed that the Wi-Fi subflow was always chosen first in each round-robin cycle. This means it is likely that the final cycle ends before the LTE subflow is used. The steady Wi-Fi packet count along with the observation that the weights are adhered to well, lead to the conclusion that when the LTE weight is set to x % of the Wi-Fi weight, a throughput increase of (nearly) x % can be achieved, assuming that the maximum throughput of the wireless connection was the bottleneck.

Furthermore, for the final experiment with weights of 200 and 140, we notice a median throughput that is 7.8 % higher than with the default MPTCP scheduler. Additionally standard deviations are slightly lower for all metrics. This configuration starts to approach the theoretical maximum MPTCP throughput, as its throughput is only 6.0 % lower than the sum of the Wi-Fi-only and LTE-only throughputs. Note that the average throughput of the LTE-only experiment was 71.2 % of the Wi-Fi-only throughput. This indicates that when the LTE weight is over 71.2 % of the Wi-Fi weight and weights are respected perfectly, the LTE interface would reach its maximum throughput and slow down the Wi-Fi interface. Overall, this shows that, when network speeds enforced by providers are the limiting factor, our WRR scheduler is a viable alternative to the default MPTCP scheduler and can even outperform it.

B. Smoothed round-trip time analysis

We also analyze the smoothed round-trip time (SRTT) as measured by the Linux kernel's TCP implementation. For every incoming non-retransmitted TCP acknowledgment (ACK), the kernel calculates how much time has passed since sending the now acknowledged segment. For the first ACK in a stream, the SRTT is simply set to that interval m. For all following ACKs, the SRTT is updated using the formula $srtt = \frac{7}{8} \times srtt + \frac{m}{8}$.

Figure 4 shows the cumulative distribution functions of the measured SRTT for the Wi-Fi only and LTE only test cases, as well as for the LTE subflow for the different scenarios with our WRR scheduler. Note that we omitted the runs with weights 2 and 30 for LTE from the figure for visibility. 90 % of all Wi-Fi SRTT samples lie between 80 ms and 155 ms, with none exceeding 255 ms. The LTE SRTT measurements are a lot worse, with all samples being fairly evenly distributed between 50 ms and 800 ms. With lower throughput on the cellular interface, the SRTT improves vastly. With LTE weights 10 and 30 for our WRR scheduler, 95 % of all SRTT samples are below 100 ms and 109 ms respectively. With LTE weight 2, this 95^{th} percentile is 120 ms. For all three of these experiments, maximum SRTT was 220 ms. The SRTT measurements start to increase considerably with LTE weight 60. While the 85^{th} percentile is only 121 ms, this increases to 184 ms for the 95^{th} percentile. The maximum measured SRTT is 520 ms. Finally, the weight 140 experiment shows a fairly linear CDF up to 303 ms for the 93^{th} percentile. Only 50 % of all samples are below 147 ms, and 95 % are below 344 ms. In this case the maximum SRTT is 699 ms. These measurements show that the mobile interface exhibits round-trip times comparable to and even lower than the Wi-Fi as long as the mobile share is kept low enough. As the round-trip time seems to increase along with the throughput for the mobile interface, the share should be kept relatively low to ensure a low RTT for the MPTCP connection. In our tests, a LTE weight 30 % of the Wi-Fi weight had no negative effect on overall RTT.

VI. Conclusion

This paper proposes an SDN-based framework for coordinated multi-path routing in heterogeneous wireless networks. This framework is located on the intersection of two active research domains: namely MPTCP and SDN. The key component is a novel MPTCP scheduler, named WRR scheduler, that is capable of dividing the segments of a TCP traffic flow across different subflows, and thus network interfaces, based on weights. These weights can be configured dynamically by a centralized controller and thus adapted depending on the current state of the network, based on real-time monitoring information. Furthermore, we present a prototype implementation of the framework with android smartphones using a Wi-Fi and LTE connection. In our evaluation we show that the traffic indeed respects the weights set by the WRR scheduler and that it is possible to achieve a higher throughput compared to the default LowRTT MPTCP scheduler, depending on the configuration of the weights. The deviation between the set LTE weight and its actual throughput share was at most 2.3 %.

References

[1] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses," Internet Requests for Comments, January 2013. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6824.txt

[2] O. Bonaventure and S. Seo, "Multipath TCP Deployments," *IETF Journal*, vol. 12, no. 2, pp. 24–27, 2016.

[3] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental evaluation of multipath tcp schedulers," in *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*. ACM, 2014, pp. 27–32.

[4] B. Arzani, A. Gurney, S. Cheng, R. Guerin, and B. T. Loo, "Impact of path characteristics and scheduling policies on mptcp performance," in *Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on*. IEEE, 2014, pp. 743–748.

[5] F. Yang, P. Amer, and N. Ekiz, "A scheduler for multipath tcp," in *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on*. IEEE, 2013, pp. 1–7.

[6] J. Hwang and J. Yoo, "Packet scheduling for multipath tcp," in *Ubiquitous and Future Networks (ICUFN), 2015 Seventh International Conference on*. IEEE, 2015, pp. 177–179.

[7] S. Ferlin, Ö. Alay, O. Mehani, and R. Boreli, "Blest: Blocking estimation-based mptcp scheduler for heterogeneous networks," in *IFIP Networking Conference (IFIP Networking) and Workshops, 2016*. IEEE, 2016, pp. 431–439.

[8] Q. De Coninck, M. Baerts, B. Hesmans, and O. Bonaventure, "A first analysis of multipath tcp on smartphones," in *International Conference on Passive and Active Network Measurement*. Springer, 2016, pp. 57–69.

[9] A. Nikravesh, Y. Guo, F. Qian, Z. M. Mao, and S. Sen, "An in-depth understanding of multipath tcp on mobile devices: Measurement and system design," in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*. ACM, 2016, pp. 189–201.

[10] K. Xu, X. Wang, W. Wei, H. Song, and B. Mao, "Toward software defined smart home," *IEEE Communications Magazine*, vol. 54, no. 5, pp. 116–122, 2016.

[11] P. Gallo, K. Kosek-Szott, S. Szott, and I. Tinnirello, "SDN@home: A method for controlling future wireless home networks," *IEEE Communications Magazine*, vol. 54, no. 5, pp. 123–131, 2016.

[12] N. Soetens, J. Famaey, M. Verstappen, and S. Latré, "SDN-based management of heterogeneous home networks," in *11th International Conference on Network and Service Management (CNSM)*, 2015, pp. 402–405.

[13] T. De Schepper, P. Bosch, E. Zeljkovic, K. De Schepper, C. Hawinkel, S. Latré, and J. Famaey, "SDN-based transparent flow scheduling for heterogeneous wireless LANs," in *International Symposium on Integrated Network Management (IM)*, 2017.

[14] R. Riggio, M. K. Marina, J. Schulz-Zander, S. Kuklinski, and T. Rasheed, "Programming abstractions for software-defined wireless networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 146–162, 2015.